

SQLskills Immersion Event

IE3: High Availability & Disaster Recovery

Module 4: Restore Internals and Procedures

Kimberly L. Tripp
Kimberly@SQLskills.com



Overview

- Phases of restore
- Recovery completion states
- Point-in-time recovery
- Restore types
- Restoring sequence and locations



2

© SQLskills, All rights reserved.
<http://www.SQLskills.com>



Phases of Restore

- **When you restore a backup, what happens? Some or all of:**
 - File creation and initialization
 - Data and/or transaction log copy
 - Redo
 - Undo
- **It depends on backup type and the restore options specified**
- **Can you improve the overall performance and minimize downtime?**
 - It depends... (more over the next few slides)

File Creation and Initialization

- **Creation of database files**
 - Files being restored must be created and initialized unless they already exist
 - Do not drop the database prior to restoring it
- **Time requirement:**
 - Depends on write throughput of I/O subsystem
 - Disks, RAID controllers, etc.
 - Same time requirement as if the database/file was being created from scratch
 - Can be SIGNIFICANTLY reduced for data files by instant initialization (which must be enabled first)
 - Remember that log files must always be zero initialized

Data and/or Transaction Log Copy

- Data and/or transaction log is copied from backup devices to data/log files
- Time requirement depends on:
 - Amount of actual data and/or log to be read, copied, and written (does not copy free space)
 - Backup compression can reduce the amount of reads
 - Performance of slowest component of the I/O subsystem
 - Disk device, backup device, controller, PCI bus, network
 - Database physical file layout and backup set stripe size
 - More files means more parallel reading and writing
- CPU usage should be insignificant unless compression is involved

Redo

- Redo phase of recovery occurs after the log has been copied
 - Reads the transaction log
 - Reapplies changes recorded in log to data pages so all pages are consistent with log
 - Performed even if WITH NORECOVERY is specified
- Time requirement depends on:
 - Amount of log to redo (affected by type and frequency of backups)
 - System performance as before

Undo

- **Undo phase of recovery occurs at the end of the restore (RECOVERY/STANDBY must process UNDO)**
 - Begins after redo reaches its target point, often the point of failure
 - Changes that were applied by active, uncommitted transactions at the target point are undone
- **Time requirement depends on:**
 - The amount of data modified by the active transactions
 - Whether or not long running transactions have to be undone
- **Planning: how long does undo take on your system?**

How to Restore

- **In an emergency, always go to the operations guide/DR handbook/run book (do you have one?)**
- **Operations guide should define:**
 - Location of backups
 - How to put together the restore sequence, depending on what is damaged
 - Commands or scripts to use
 - Amount of time expected per step during restore process
- **Recommended to NOT use SSMS when recovering from disaster: you want to see what's happening**
- **Make sure to use the correct RESTORE options**
- **Use log shipping for automation/testing**

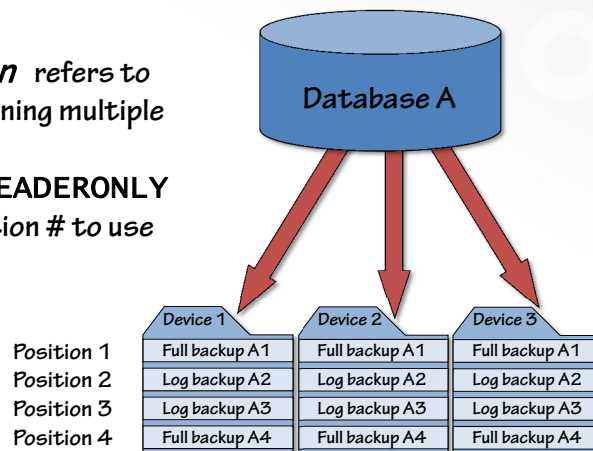
Restore Options

- **RESTORE LABELONLY**
 - Information on media set that device is part of
- **RESTORE HEADERONLY**
 - Information on all backup sets in the backup file(s)
- **RESTORE FILELISTONLY**
 - List of database files contained within a backup
- **RESTORE VERIFYONLY**
 - Discussed during backup and used to verify the integrity of the backup
- Preferably have scripts that extract information from the backup history tables in msdb

Restoring from Multi-File Backups

WITH FILE = *n* refers to backup set containing multiple backups

Use **RESTORE HEADERONLY** to find which position # to use

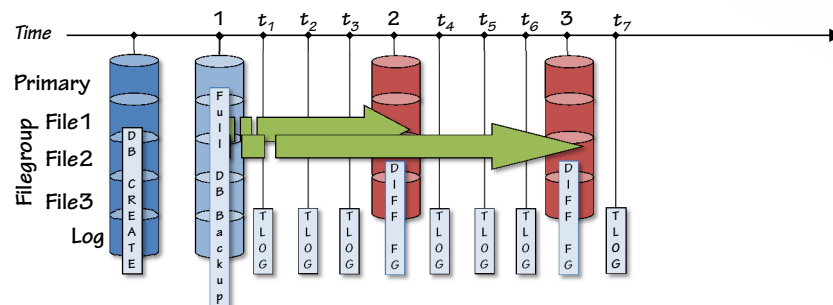


Demo

Restoring from multi-file backups

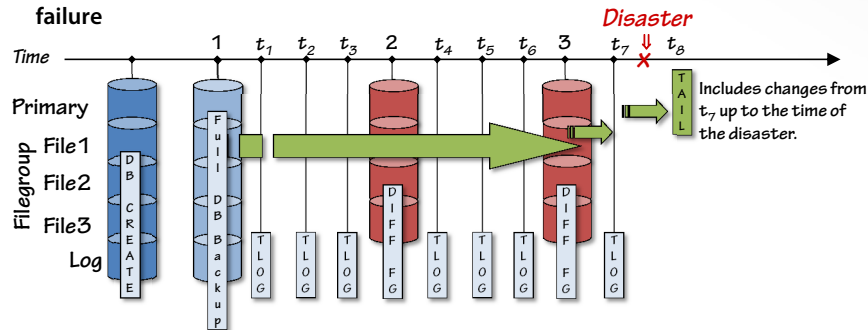
Differential Backup Reminder

- Differentials are since the last full...
- Always want to restore the fewest backups possible to reduce downtime

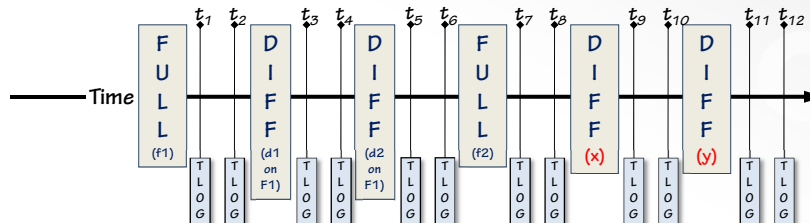


Recovery with Full, Diff, Log

- Backup 1 is full
- Backups 2, 3 are differential
- In the event of failure at time x
- Backup "tail" of log
- Restore full (1), last differential (3) and all log backups up to the point of the failure



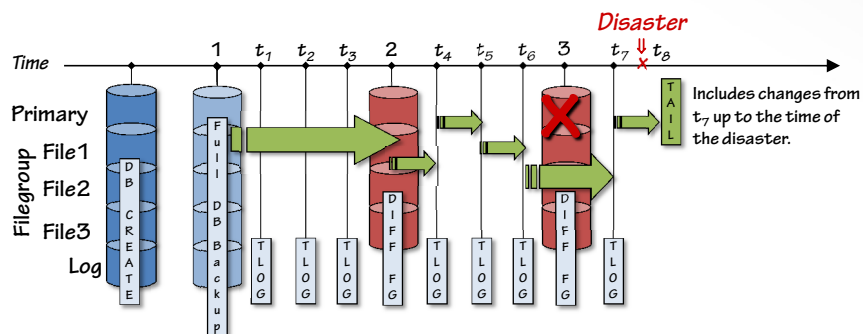
Recovery Sequence with/without COPY_ONLY



- If the full database backup at f2 is performed without using COPY_ONLY then
 - Recover the full at f2 then the diff at y then t₁₁ and t₁₂
 - What if the backup at f2 was deleted/thrown away?
 - Recover the full at f1 then the diff at d2 then all of the tlogs t₅ through t₁₂
- If the full database backup at f2 is performed using COPY_ONLY, then
 - Recover the full at f1 then the diff at y then t₁₁ and t₁₂

Falling Back on Log Backups

- What if differential backup 3 was bad?
- Restore full (1), latest working differential (2) and all log backups up to the point of the failure (t4 through t7) and finally the tail of the log (t8)
- Begs question of how long to keep backups?



Continuity of the Log Backup Chain

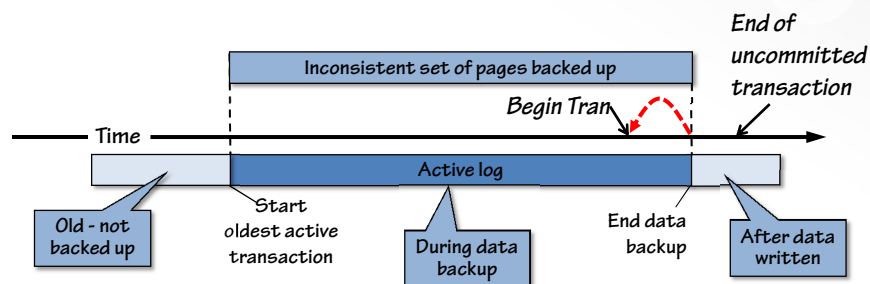
- There must be an unbroken series of backups from the previous full backup to the point at which the restore sequence ends
- If there are time periods that are not covered by differential or log backups, the restore sequence cannot proceed
- Switching to SIMPLE breaks the log backup chain
 - Must take new full or diff backup to restart
- Dumping the log breaks the log backup chain
 - Must take new full or diff backup to restart
- Beware of using file/filegroup backups after SIMPLE

Recovery Completion State: RECOVERY

- **RESTORE ... WITH RECOVERY**
 - This is the default – catches lots of people out
- Using **WITH RECOVERY** leaves the database operational and no additional backups (differential or log) can be restored
- **Active transactions at the time the backup completed are rolled back**
 - I.e. the UNDO portion of recovery is run, which generates log records to affect the rolling back of active transactions
 - As the database has generated new transaction log records, no additional backups can be restored
- **The restored database is left in a transactionally consistent state and the restore sequence is completed**

Recovery Completion State: RECOVERY

- **RESTORE ... WITH RECOVERY**



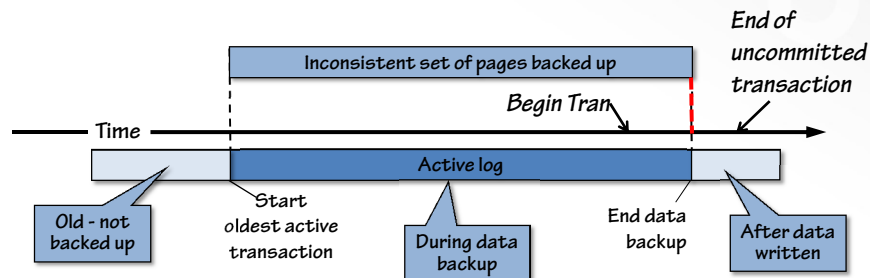
- The transaction is still active when the backup completes so the UNDO phase of restore will rollback the transaction in the restored copy of the database

Recovery Completion State: NORECOVERY

- **RESTORE ... WITH NORECOVERY**
- Using WITH NORECOVERY does not leave the database operational and additional backups (differential or log) can be restored
- Active transactions at the time the backup completed are not rolled back
- The database remains inaccessible and in the RESTORING state until recovery is manually completed or another backup is restored
- The REDO part of recovery will have been performed
 - Slightly confusing, but REDO is done on each backup that is restored as it does not affect whether more backups can be restored

Recovery Completion State: NORECOVERY

- **RESTORE ... WITH NORECOVERY**



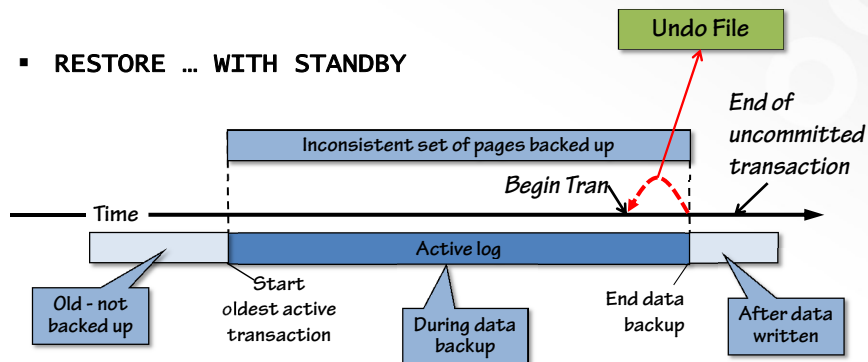
- Although the transaction is still active when the backup completes, the UNDO phase of recovery is not run by restore so the log records remain
- The next log backup restored may include the commit

Recovery Completion State: STANDBY

- **RESTORE ... WITH STANDBY=<undo file name>**
- Using STANDBY leaves the database read-only and additional transaction logs can be restored
- **Active transactions at the time the backup completed are rolled back but the log records necessary to do it are written to a file to allow further restores**
 - I.e. the UNDO part of recovery is run, but without affecting the transaction log itself in the restoring database
 - This means that the next restore will undo the UNDO, and then continue as if the previous restore had been performed using WITH NORECOVERY
- **Very useful during a long, complex restore sequence to see what state the database/data is in**

Recovery Completion State: STANDBY

- **RESTORE ... WITH STANDBY**



- The transaction is still active when the backup completes so the UNDO phase of restore will rollback the transaction in the restored copy of the database, but into an undo file
- The next backup will 'undo the undo' before continuing

Recovery Completion State

Summary of Important Points

- RECOVERY is the default
- Use NORECOVERY for safety and make it your default for all restore operations in the restore sequence
 - If you make a mistake, you're back to square-one
 - Required when setting up database mirroring or log shipping on the initial restore of the full database backup
- Use RESTORE DATABASE ... WITH RECOVERY to complete the restore sequence
- STANDBY uses the UNDO file to save info from transactions that are pending at the end of the restore

Restoring an Entire Database to a Point in Time

- Common example: restore a database to the state it was in before a user or application error occurred (e.g. dropping a table)
- Start with most recent full backup and proceed with smallest set of backups to the desired point
- Best practice is to use WITH STOPAT on all operations in the restore sequence
 - Date and time specific, or
 - Transaction name (i.e. marked transaction)
 - LSN
 - Demo on Friday
- Note: All work after the stop point is lost

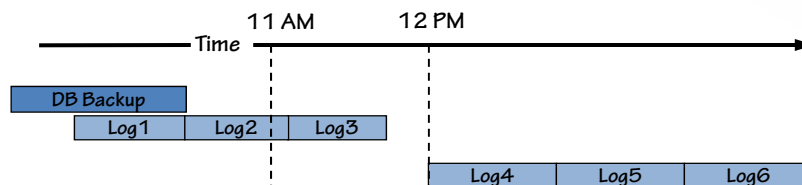
Demo

Recovery options on RESTORE and point-in-time recovery

Continuity of the Log Chain in P.I.T.

- Updating a database after point-in-time restore creates a new recovery path
- Follow point-in-time restore with a new full backup to avoid having to point-in-time restore again if another disaster occurs

At 12 noon, the database is restored and recovered to 11:00 AM



- The log backups taken after the database was recovered are only useful if exactly the same recovery path is followed
 - Log backup #3 ends the first recovery path

Piecemeal Restore

Restoring from Scratch

- **Example:** You want to recover deleted data from a table but don't want to restore the entire database
- **You can restore a subset of the filegroups to create a new, smaller database in Enterprise Edition**
 - First restore (of the primary filegroup) must use the PARTIAL option
- **Manually extract deleted or damaged data from new database, then merge or insert it into production database**
 - Merge/insert is time-consuming/error-prone
- **Not the same as a piecemeal restore into an existing database (see next slide)**

Demo

Using WITH PARTIAL

Piecemeal Restore

Restoring into Existing Database

- **Do not have to restore the entire database as long as the right backups exist**
 - Page-level, file-level, filegroup-level restore
- **Log backups must exist to allow the restoring portion of the database to be brought in sync with the rest of the database**
 - The sync point is the time at which the piecemeal restore started (as nothing could affect that portion of the database afterwards)
- **For file/filegroup restore, database must have been designed to allow this**
- **Enterprise Edition allows the rest of the database to be online while the restore is taking place**
 - Online piecemeal restore and partial database availability
- **Page-level restore demos in Module 10**

Identifying Log Backups to Apply

- **In a complex restore sequence, there may be differential and log backups that cover the same time interval**
- **No point restoring unneeded log backups, but also must make sure that the log backup covers the right time period (i.e. set of LSNs in the log)**
 - SQL Server will tell you if either is not the case
- **Finding the minimum required LSN or min-LSN can be done using:**
 - Backup history tables in msdb
 - Output from RESTORE HEADERONLY
- **Make sure you're backing up msdb (and other system databases too)**

Finding Effective Min LSN

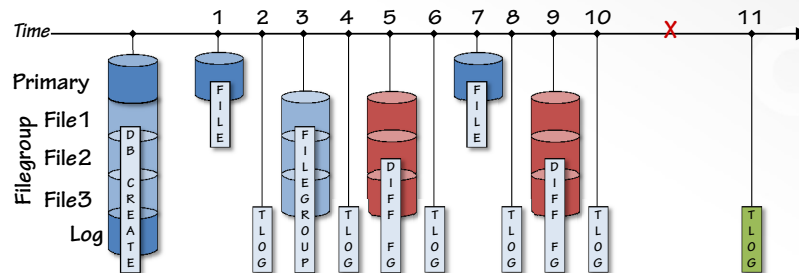
- From msdb, if the information exists

```
SELECT Backup_Start_Date,  
       [Name],  
       [Description],  
       First_LSN,  
       Last_LSN,  
       Backup_Finish_Date,  
       * -- OR Just *  
FROM msdb.dbo.backupset AS s  
     JOIN msdb.dbo.backupmediafamily AS m  
         ON s.media_set_id = m.media_set_id  
WHERE database_name = 'databasename'  
ORDER BY 1 ASC
```

Demo

Finding the min LSN

Case Study (1)

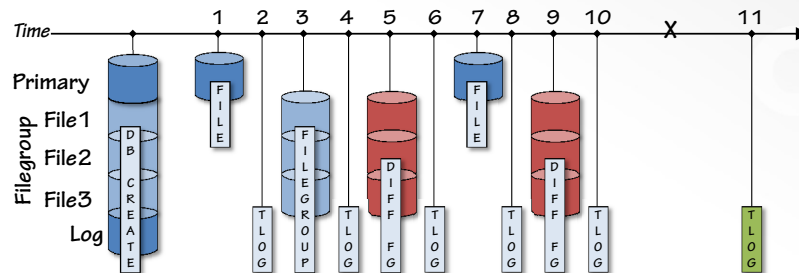


- Database with 5 files, two filegroups: primary, FG
- Point in time 1 and 7 = file backups
- Point in time 3 = filegroup backup
- Point in time 5 and 9 = filegroup differential backups
- Point in time 2, 4, 6, 8 and 10 = transaction log backups
- Point in time 11 = transaction log tail backup

Case Study (2)

- Step 1: log tail backup
- Step 2: determine how to get the structure of the database created
 - Which backups are for which parts of the database?
- Step 3: bring the file/filegroups up to the minute as fast as possible
 - Are there any differentials that can short-circuit log restores?
- Step 4: apply the transaction logs that will make this database consistent
 - Find the min LSN from the oldest full/diff backup from steps 2 and 3

Case Study (3)



▪ **Sequence is:**

- 1) Take log backup #11 (tail of the log)
- 2) Restore #7 (last full file backup of primary file)
- 3) Restore #3 (last full file backup of secondary filegroup)
- 4) Restore #9 (most recent diff backup of secondary filegroup)
- 5) Restore #8 (log that might affect the primary file)
- 6) Restore #10 (next log in the chain)
- 7) Restore #11 (log tail, final log in the chain)

VLDB Concerns when Restoring

- **Restoring whole database can take a lot of time**
- **Media failure**
 - If isolated corruption can restore the file or filegroup from a full backup
 - Must go up-to-the-point-of-the-crash and only if piecemeal restore is an option
- **User error**
 - Best choice is a secondary server that's not quite up to date, e.g. log shipping with a load delay!
- **VLDB backup and restore is very challenging**
 - Do as much as possible to speed things up
 - Compression, striping, partitioning, granular backups

SQL Server Management Studio

Restore/Recovery

- Database Recovery Advisor
- Restore plans – including more complicated forks – are supported
- Point-in-time recovery using a visual timeline
- Page restore UI (Paul covers page restores)

Demo

Using 2012 SSMS for complex recovery sequences

Restarting a Restore Operation

- Restore goes through four phases
 - Create and initialize files
 - Copy data and/or log
 - REDO
 - UNDO
- If an operation is interrupted, restore writes a checkpoint file (NOT anything to do with database checkpoints) to the instance BACKUP directory
- Restore operation can be started with RESTORE option to resume from last completed phase
- Can save many hours of repeated operations

Demo

Using WITH RESTORE

Restore Sequence: Media Failure

- Backup the tail of the log
- Identify damaged devices (i.e. what needs to be restored?)
- Verify backups (location and existence)
- Provision undamaged location to restore to
- Decide the SMALLEST restore possible: page, file, filegroup, multiple filegroups, database...
- Restore (all WITH NORECOVERY):
 - Full database backup
 - Optionally, restore differentials
 - All required log backups
 - Tail-of-the-log backup
- Finish recovering the database
 - RESTORE DATABASE ... WITH RECOVERY

Restore Sequence: Human Error

- Investigate nature of problem: do you know what the error was, and when it occurred?
 - Excellent if you do know, not so good if you don't
 - Are you auditing? Can you look in audit logs? Default trace?
 - Can you restore to other location and attempt to find when data changed?
 - PAINFULLY time consuming
 - RESTORE ... WITH STANDBY and slowly query the database
 - Third-party products can help (but beware, not infallible!)
 - E.g. Apex SQL Log
- Possibly take damaged component offline
- Choose a restore location
 - In place or alternate location
- Begin restore sequence as for media failure
- Possibly merge in damaged data to existing database

Restore in Place

- All application dependent information is already on the server (i.e. the application ecosystem)
- Replaces current database
- Must replace damaged devices first or use WITH MOVE to restore on stable media
- Restore based on backup strategy
- No further steps necessary as everything required is already there
- Easiest restore location by far!

Restoring to Alternate Location

- **Same instance, different name – relatively easy!**
 - Consider name changes to jobs and SQLIS packages, and anything that connects directly
 - Client connectivity issues
- **Different instance on same server (includes all of the above)**
 - Different logins
 - Different msdb (Agent Jobs/SSIS packages)
 - Different service accounts (mail profiles)
 - Different master (user-defined SPs do not exist)
 - Different servername (client connectivity)
- **Physically different server (includes all of the above)**
 - Any external dependencies? (batch files, email...)
 - Local Windows authentication accounts may not exist
- KB 246133 on migrating logins between servers

Restoring to Non-Enterprise

- SQL Server 2005 will not allow a database that contains table/index partitioning to restore on any Edition except Enterprise and Developer
- SQL Server 2008 expands this list to also include:
 - Change data capture
 - Data compression
 - Transparent data encryption
- All of these require elevated privileges except data compression, which only requires ALTER TABLE
- This can be disastrous in a disaster recovery situation as the database must be restored BEFORE the server can tell whether any of these are present
- Use sys.dm_db_persisted_sku_features to check

Restoring System Databases (1)

- System databases can only restore from backups of the same SP level as the server
 - E.g. an RTM backup of msdb will not restore on SP1
- model
 - Restore in same way as for user databases
- msdb
 - Restore in same way as for user databases
 - Stop SQL Server Agent before restoring
 - Make sure to set the recovery model to FULL if msdb was rebuilt
- resourcedb
 - Cannot be backed up or restored
 - Use file-system copy operations but be careful not to overwrite resourcedb with an older version
- tempdb
 - Cannot be backed up or restored

Restoring System Databases (2)

- **Restoring master**
 - Start SQL Server in single-user mode with the -m startup parameter
 - Restore from backup, using WITH REPLACE
 - Server shuts down automatically, so remove -m and restart
 - For any changes after the master backup was made:
 - Reattach any new databases, Recreate any new users/logins, change any configuration settings, etc. (should ALWAYS backup master after anything in the "system configuration" changes!!)
- **Rebuilding master**
 - Necessary if no backup exists or the instance will not start
 - If you have to rebuild master to allow SQL Server to start, all data is lost
 - Restore master if possible using steps above
 - For any changes after the master rebuild was performed:
 - Reattach any new databases
 - Recreate any new users/logins

Data Encryption

- **When restoring a database that contains encrypted data there are multiple possibilities:**
 - Same server/instance – no troubles
 - Different instance or different server – different service master key
 - Must open up the database master key and RE-encrypt the data using the new instance's service master key – EASY if you know to do it and you have the password with which the original database master key was created
- **Especially important in SQL Server 2008 with Transparent Data Encryption**
 - Encrypted database cannot be restored without the encryption key

Review

- Phases of restore
- Recovery completion states
- Point-in-time recovery
- Restore types
- Restoring sequence and locations

Resources

- **Articles:**
 - TechNet Magazine: Recovering from Disasters using Backups
 - <http://technet.microsoft.com/en-us/magazine/ee677581.aspx>
 - SQL Server Magazine: Advanced BACKUP and RESTORE Options
 - <http://tinyurl.com/6ykdrnf>
- **Blog posts:**
 - <http://www.sqlskills.com/blogs/paul/category/BackupRestore.aspx>

Questions?

